Note: In this document we will be using the term "IRDB" as a short alias for "InMemory.Net".

Using IRDB in a Dot Net Project

ODBC Driver

A 32-bit odbc driver is installed as part of the server installation. This can be used to connect to an IRDB server. This provides the simplest and most generic way of connecting to an IRDB server.

IRDBProvider.DLL

A simple dot net provider is provided for IRDB. You need to add IRDBProvider.dll and IRDB.DLL to your project. This allows you to use familiar Dot Net API's for communicating with the IRDB Server. Please see the IRDBProvider detailed examples for more detailed examples and how to use the IRDBProvider.

IRDB.DLL

Importing Data, Saving Data, and Executing Queries in Process .

Using Visual Studio 2010 or higher, add irdb.dll as a reference to your project. Irdb.dll can generally be found in c:\irdb\irdb.dll. This may change depending on your installation location. The IRDB.dll is compiled as ANY Cpu, so should be able to support 32 or 64 bit builds. The IRDB.dll also has dependencies on System.Numerics, Antlr4.Runtime.net40.dll, and Interop.ADODB.dll

IRDB can also be used from an ASP.Net 4.0 app (or higher) by making sure the irdb.dll files are available to the page.

Depending on which method you are using you will need to add the following to your Using clause. For the example here we are using C#.

```
using irdb;
using System.Data;
using System.Data.Odbc;
using System.Data.SqlClient;
using System.Data.OleDb;
using System.Data.Common;
```

Not all the System.Data may be required. If you are only using the SqlClient, then you may not need to include ODBC or OleDB. If you are using a different database Native Client, you may also need to include that.

In the next example we are going to write a program to connect to a sql database, import a few tables into an IRDB database, save the database to disk, load it into a new database and then Execute a query against it.

The following Creates a Connection to a Sql Server DB using SQL authentication and the Sql Native Client.

```
String connString = @"Data Source=yourDatabaseServerIpORName;Initial
Catalog=DBNameOnServer;User Id=yourUserName;Password=YourPwd;";
SqlConnection conn = new SqlConnection (connString);
conn.Open();
```

Here we create a new InMemoryDatabase using the InMemoryDatabase Class

```
InMemoryDatabase db = new InMemoryDatabase();
```

This shows how to pass a database connection and a connection to the InMemoryDatabase, then add it to the in memory database.

```
String sql = "Select * from ir_wiprodprofit";
db.add("ir_wiprodprofit" , conn, sql);
```

Saving the InMemoryDatabase to a file..

```
db.save(@"c:\irdb\test1.irdb");
```

This shows how to load an InMemoryDatabase back in again.

```
InMemoryDatabase db2 = new InMemoryDatabase(@"c:\irdb\test1.irdb");
```

How to take an SQL Statement and execute it against the InMemoryDatabase.

```
String sql = @"SELECT SUM(unbilled_wip) FROM ir_wiprodprofit";
InMemoryTable inMemoryTable = db.execute(sql, out error, out
debugText);
if (error.Length > 0 || inMemoryTable == null){
      Console.WriteLine(error);
```

```
        Console.WriteLine("Failure");
}
else{
        Console.WriteLine("Success");
}
```

The following takes the resulting InMemoryTable and converts it to a Standard Dot Net datatable that can then be used across most other Dot net Libraries

```
DataTable dataTable = inMemoryTable.toDataTable();
```

Convert the result to an ADODB.Recordset..

```
ADODB.Recordset rsResults = new ADODB.Recordset();
inMemoryTable.ConvertToRecordset(rsResults);
```

The following static method will show how to display your timings.

```
InMemoryDatabase.DisplayTimings();
```

Add the resulting table to the InMemory Database

```
db.add("new_table_name_to_add" , inMemoryTable);
```

This shows how to remove a table from the InMemory database

```
Bool success=db.removeTable("table_to_remove");
```

InMemory Database Class Reference

For Importing data, saving/ loading databases and executing SQL in process, here is the standard reference.

Constructors for InMemoryDatabase

```
new InMemoryDatabase();
```

Construct a Blank InMemoryDatabase.

```
new InMemoryDatabase(String fileName);
```

Import a table to the InMemoryDatabase given a DBConnection and SQL statement.

```
add ( String tableNameToAddTo, DBConnection connection, String sql )
```

This is the main method for importing tables into IRDB.
`tableNameToAddTo` *is the name of the tablename to save it as.*
`DBConnection` *`connection` can be either an ODBC Connection, OLEDB Connection, SQL Connection or any other Connection that implements DBConnection from System.Data. The connection should be opened.*
`String sql` *is the SQL that you would like to execute against the connection, to import into the in memory database.*

IRDB currently displays a **.** for every 100,000 records imported.

Adding an InMemoryTable to an InMemoryDatabase

```
add ( String tableNameToAddTo, InMemoryTable inMemoryTable )
```

`tableNameToAddTo` *is the name of the tablename to save it as in the in memory database.*
`inMemoryTable` *is an In Memory Table return from a prior execute command*

Remove a table from an InMemoryDatabase.

```
removeTable ( String tableNameToRemove )
```

`tableNameToRemove` *is the name of the table name to remove from the In Memory Database.*

Save the InMemoryDatabase

```
save ( String filename )
```

`filename` *is the name of the file that it is saved to.*

Run/ Execute an SQL query

```
execute(String sql, out String error, out String debugText);
```

This takes a SQL statement passed in the first argument and executes it against the current IRDB database. It also takes 2 other blank string arguments. These 2 strings need to be passed with the `out` prefix as they may return information. Any error is returned in the second parameter, and the third parameter may contain useful debugging information.

This method returns an InMemoryTable Object, with the results in a columnar in memory table. This is the same object used internally to store tables. If the table is null or the error string contains text, then there was a problem executing the query.

If the returned InMemoryTable Object is null or the length of errors is Non Zero it means there was a problem executing your query.

What can be done with the resulting table?

From an interaction point of view there are 3 main things that can be done with the resulting table. You can convert add it back to the in memory database using the `add` method. You can also convert it to a data table (by calling `toDataTable() )` or to a ADODB.Recordset (by calling `ConvertToRecordset(rsResults))` where rsResults is a newly created blank ADODB.Recordset.

Connecting to the IRDB Server Process from a Dot Net Application irdb.IPClient

Add the irdb.dll as a reference to your project, then add

```
using irdb;
```

to your using section.

Communicating with the IRDB Server

The IPClient class is used to communicate with the IRDB Server. IPClient cannot make any changes to an IRDB database, and is essentially Read Only.

```
IPClient(String host, int port,String username,String password, out bool success)
```

The constructor for IPClient is as follows. Host is the name or IP address of the server. Use localhost for an IRDB on the same computer. The port is normally 5060. Specify a username. The username is a placeholder at the moment, and support will be added in a later version. The password is required as well to authenticate against the IRDB server. The resulting success returns whether it was able to successfully connect or not.

Load a Remote Database

To have a remote Server load the specified database saved on the Server system...

```
bool load ( String dbName )
```

`dbName` is the name of the irdb file in the data dir without extension, on the irdb server. Return type `bool` of indicates success/failure

Reload a Remote Database

To reload a remote database use the following syntax.

```
public bool reload(String dbName, out String errors)
```

This will happen as a background process, and will replace the currently loaded database once complete. If the database file is already reloading, it will not trigger another reload. If the reload takes a long time, it will return the following error:

```
        Database is currently Loading on the Server. Please try back again.
```

Unload a Remote Database

Unload a specified database from a remote server with the following...

```
public bool unload(String dbName, out String errors)
```

This helps reduce resource usage on the remote server.

Check the Load-State of a Remote Database

```
public int isLoading(String dbName)
```

Returns an int as follows. `ERROR=-1, NOT_LOADED=0, LOADING=1,LOADED=2`

Close Remote Server Connection

```
void close()
```

Closes the connection to the remote server.

Execute an SQL Statement against a Remote Server

It is possible to execute an sql statement against a remote database. This will also cause the database to be loaded. Null table or errors in third parameter passed as an `out` parameter indicate an error occurred. The InMemoryTable can then be converted to a datatable or ADODB.Recordset.

```
InMemoryTable execute  ( String dbName , String sql, out String
errors )
```

`dbName` is the Database name
`sql` is the SQL statement.

Execute an SQL Statement against a Remote Server with parameters

```
InMemoryTable executeWithParams(String dbName, String sql, Dictionary<String, ImpVar>
parameters, out String errors)
```

Same as previous function but you can pass a list of named variables in the dictionary, that will appear as variables in the sql statement. This can also be used to parameterize the where clause.

List Tables in a Remote Database

To get a list of tables in a remote database, use the following syntax:

```
Object[] listTables(String dbName)
```

This passes back an Object array of the list of tables specified in the database called `dbName` on the IRDB server. This Object[] Is really a String[], but because of the way COM mangles Dot Net strings, we use an Object[].

List Databases on a Remote Server

To list databases on a remote sever, use:

```
Object[] listDatabases()
```

This passes back an Object array of the list of the databases on the IRDB server. This Object[] Is really a String[], but because of the way COM mangling Dot Net strings, we use an Object[].

```
DBStats[] listDatabasesDetailed()
```

This function returns back a more detailed structure with more information about what databases are loaded.
DBstats is defined as follows

```
public class DBStats
    {
        public string dbName;
        public LoadStatus loadStatus;
        public long estimatedSize=0;
        public DateTime loadedTime;
        public DateTime loadedFileCreated ;
        public long loadedFileSize;
        public DateTime currentFileCreated ;
        public long currentFileSize ;
    }
```

List Fields in a given Table in a Remote Database

```
public Object[] listFields(String dbName, String tableName, List<DataTypes> datatypes)
```

This passes back an Object array of the list of fields in `tableName` in database `dbName` on the IRDB server. This Object[] Is really a String[], but because of the way COM mangling Dot Net strings, we use an Object[]. `Datatypes` should be blank list, and it will contain the corresponding datatype. This function is designed for analysing metadata.

Create a Server Side Cursor, execute against it and drop cursors.

```
public InMemoryTable createCursor(String dbName, String sql, out String cursorName, 
out String errors)
public InMemoryTable fetchFromCursor(String databaseName, String cursorName, int 
startRow, int endRow)
public bool closeCursor(String databaseName, String cursorName, out String errors)
```

The above methods are used to create a server side cursor, and then retrieve the data in chunks from the cursor and then close it.

Create a Server Side Statement, execute against it (with cursors) and drop statements.

```
public InMemoryTable prepareStatement(String dbName, String sql, out String 
statementName, out int noParams, out String errors)
public bool dropStatement(String dbName, String statementName, out String errors)
public InMemoryTable statementExecute(String dbName, String statementName, 
List<String> parameters, out String errors)
public InMemoryTable statementCreateCursor(String dbName, String statementName, 
List<String> parameters, out String cursorName, out String errors)
```

The above methods allow you to create server side statements, that can take a list of parameters, that are then substituted into the statement, and return either direct results or a result in cursor format. Useful for writing ODBC Drivers! Statements are SQL type statements that have ? in them for parameters. The list of string parameters will be converted and parsed as the appropriate data type, based in general on analysing the other side of the expression in the where clause.

COM Client

The Com Client is installed as part of the IRDB Server Installation. Currently the COM Client supports the same core functionality for connecting to the IRDB Server as the IPClient. All the method for connecting to a remote IRDB Server start with `remote`. The name of the Com Client class is `"irdb.irdbcom"`

```
Object [] remoteListTables(String dbName)
Object[] remoteListFields(String dbName,String tableName)
void remoteDisconnect()
```

```
ADODB.Recordset remoteExecute(String dbName, String sql, out String
errors)
bool remoteLoad(String dbName)
bool remoteReload(String dbName, out String errors)
bool remoteUnloadDatabase(String dbName)
bool remoteConnect(String conn_string)
```

Connection string (`conn_string`*) has 5 components separated by ';'*
irdb = name of remote database.
username = name of user to use to connect to. A value of default is used, when not specified.
pwd = password to connect
host = name of remote server. Dns or IPAddress
port = remote port. Usually 5060.

eg. *irdb=northwind;pwd=mypassword*
irdb=northwind;host=192.168.0.22;pwd=mypassword
irdb=northwind;host=192.168.0.22;username=default;pwd=mypassword

```
bool remoteConnect(String host, int port,String username, String password, bool
autotimeout)
int remoteIsLoading(String dbName, out String errors)
void suppressExceptions()
```

`suppressExceptions` can be used to stop the remote API part of the COM Client from throwing Exceptions.

Here is a small sample VBS Scripting code to using the above API

```
set conn=createobject("irdb.irdbcom")
success=conn.remoteconnect ("irdb=test1;pwd=!TopSecret")
success=conn.remoteLoad("northwind")
tables = conn.remoteListTables("northwind")
msgbox "Found " & ubound(tables)+1 & " tables in
northwind.irdb"
conn.remotedisconnect
```

It is also able to Create a new Database locally, and import tables into it, and then save it again locally.

However we now recommend using the IRDBImport/IRDBImport32 EXE programs for generating IRDBs

All the Following will take the current COM Object, treat it as an In Memory Database, and allow you to add tables , save and load data.

`dsn` is a database connection string that is passed to create the underlying Dot Net OleDBConnection SqlConnection and OdbcConnection respectively.

```
void addOledbTable(String dsn, String sourceSQL, String tableName)
void addSqlServerTable(String dsn, String sourceSQL, String
tableName)
void addOdbcTable(String dsn, String sourceSQL, String tableName)
bool save(String fileName)
void close()
void load(String fileName)
```

IRDBProvider.DLL Detailed Examples

Example program to connect using the IRDBProvider, and get a DbDataReader and move it into a DataTable

```
public static void testDataset()
        {
            IRDBConnection conn = new IRDBConnection();
            conn.ConnectionString =
"irdb=northwind;host=localhost;port=5060;pwd=!TopSecret";
            conn.Open();

            DbCommand cmd = conn.CreateCommand();

            cmd.CommandText = "Select * from orders";

            DbDataReader reader = cmd.ExecuteReader();

            DataTable dt = new DataTable();
            dt.Load(reader);

            conn.Close();

            Console.WriteLine(dt.Rows.Count + " rows in table.");

        }
```

Example program to connect using the IRDBProvider with parameters.

```
public static void paramTest()
        {
            IRDBConnection conn = new IRDBConnection();
            conn.ConnectionString =
"irdb=northwind;host=localhost;port=5060;pwd=!TopSecret";
            conn.Open();

            DbCommand cmd = conn.CreateCommand();

            cmd.CommandText = "Select * from orders where orderid = @orderid or
orderid = @orderid2 or orderid = @orderid3";

            cmd.Parameters.Add(new IRDBParameter("@orderid", 10248));
            cmd.Parameters.Add(new IRDBParameter("@orderid2", 10249));

            DbParameter dbParam = cmd.CreateParameter();
            dbParam.DbType = DbType.Int32;  // Explicitly setting data type is
optional
            dbParam.ParameterName = "@orderid3";
            dbParam.Value = 10250.88;
            cmd.Parameters.Add(dbParam);


            DbDataReader reader = cmd.ExecuteReader();
            DataTable dt = new DataTable();
            dt.Load(reader);

            conn.Close();

            Console.WriteLine(dt.Rows.Count + " rows in table.");
        }
```

Example program to connect using the IRDBProvider and iterate through a DbDataReader.

```
public static void testIteraction()
        {
            IRDBConnection conn = new IRDBConnection();
            conn.ConnectionString =
"irdb=northwind;host=localhost;port=5060;pwd=!TopSecret";
            conn.Open();

            DbCommand cmd = conn.CreateCommand();

            cmd.CommandText = "Select *,cast(freight as decimal) as freightd from
orders";

            DbDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                Console.WriteLine(reader["orderid"] + " " + reader["orderdate"] + " "
+ reader["customerid"] + " " + reader["freight"] + " " + reader["freightd"] + " " +
reader[0]);
                Console.WriteLine(reader.GetDecimal(0));
            }
            conn.Close();
 }
```

Example program to connect using the IRDBProvider and Fill a Dataset using a DataAdapter

```
public static void test22a()
        {
            IRDBConnection conn = new IRDBConnection();
            conn.ConnectionString =
"irdb=northwind;host=localhost;port=5060;pwd=!TopSecret";
            conn.Open();

            DbCommand cmd = conn.CreateCommand();

            cmd.CommandText = "Select * from orders";

            IDbDataAdapter dataAdapter = new IRDBDataAdapter(cmd);
            DataSet ds = new DataSet();
            dataAdapter.Fill(ds);
            conn.Close();
            DataTable dt = ds.Tables[0];
            Console.WriteLine("No of Rows Imported=" + dt.Rows.Count);
        }
```

Example program to connect using the IRDBProvider and list the databases.

```
public static void testListDatabases()
        {

            IRDBConnection conn = new IRDBConnection();
            conn.ConnectionString = "host=localhost;port=5060;pwd=!TopSecret";
            conn.Open();
            DataTable table = conn.GetSchema("databases", null);

            foreach (DataRow dr in table.Rows)
            {
                Console.WriteLine(dr["databasename"]);
            }

            conn.Close();
        }
```

Call a Server Side Stored Procedure with Parameters

```
public static void paramTestSP()
        {
            IRDBConnection conn = new IRDBConnection();
            conn.ConnectionString =
"irdb=cch;host=localhost;port=5060;pwd=!TopSecret";
            conn.Open();
            DbCommand cmd = conn.CreateCommand();

            cmd.CommandType = CommandType.StoredProcedure;
            cmd.CommandText = "LookupOrder";
            cmd.Parameters.Add(new IRDBParameter("@param1", 335));
            DbDataReader reader = cmd.ExecuteReader();

            DataTable dt = new DataTable();
            dt.Load(reader);

            conn.Close();
            Console.WriteLine(dt.Rows.Count + " rows in table.");
        }
```

Assuming you have a stored procedure called LookupOrder that takes one parameter called @param1

General Performance Observations

Currently InMemory.Net can be executed in 32bit mode, but you will not be able to generate as large a dataset as you would in 64bit mode.

In our testing we found running in a 64bit environment much more scalable than a 32bit environment. In our test environment in 32bit, we were able to use an effective in memory database size after compression of about 600-700 Mbs in converting.

32bit dot net processes are limited to using about 1.4-1.5 Gbs of Memory.

We found that due to LOH fragmentation, memory used to buffer rows during loading, wasn’t immediately released during the columnisation process. This corresponded to about 18 million rows of our test table structure.

The 64 bit environment was able to convert about 180 million rows with 8 GB of ram. And about 400 million rows using 16GB of ram. The 64 bit environment didn’t seem to suffer from LOH fragmentation as Pages could be reused by the virtual paging system.

The Com Client can be used from any Microsoft programming environment that supports COM. This includes VBS Script files, Excel, Access, Classic ASP.

The Com Client is used by IR3.5 to communicate with the IRDB Server.

The 32 bit ODBC driver should be usable from all applications that support ODBC.